

TEMPORAL ENSEMBLING FOR SEMI-SUPERVISED LEARNING

Samuli Laine

NVIDIA

slaine@nvidia.com

Timo Aila

NVIDIA

taila@nvidia.com

ABSTRACT

In this paper, we present a simple and efficient method for training deep neural networks in a semi-supervised setting where only a small portion of training data is labeled. We introduce self-ensembling, where we form a consensus prediction of the unknown labels using the outputs of the network-in-training on different epochs, and most importantly, under different regularization and input augmentation conditions. This ensemble prediction can be expected to be a better predictor for the unknown labels than the output of the network at the most recent training epoch, and can thus be used as a target for training. Using our method, we set new records for two standard semi-supervised learning benchmarks, reducing the (non-augmented) classification error rate from 18.44% to 7.05% in SVHN with 500 labels and from 18.63% to 16.55% in CIFAR-10 with 4000 labels, and further to 5.12% and 12.16% by enabling the standard augmentations. We additionally demonstrate good tolerance to incorrect labels.

1 INTRODUCTION

It has long been known that an ensemble of multiple neural networks generally yields better predictions than a single network in the ensemble. This effect has also been indirectly exploited when training a single network through dropout (Srivastava et al., 2014), dropconnect (Wan et al., 2013), or stochastic depth (Huang et al., 2016) regularization methods, and in swapout networks (Singh et al., 2016), where training always focuses on a particular subset of the network, and thus the complete network can be seen as an implicit ensemble of such trained sub-networks. We extend this idea by forming ensemble predictions during training, using the outputs of a single network on different training epochs and under different regularization and input augmentation conditions. Our training still operates on a single network, but the predictions made on different epochs correspond to an ensemble prediction of a large number of individual sub-networks because of dropout regularization.

This ensemble prediction can be exploited for semi-supervised learning where only a small portion of training data is labeled. If we compare the ensemble prediction to the current output of the network being trained, the ensemble prediction is likely to be closer to the correct, unknown labels of the unlabeled inputs. Therefore the labels inferred this way can be used as training targets for the unlabeled inputs. Our method relies heavily on dropout regularization and versatile input augmentation. Indeed, without neither, there would be much less reason to place confidence in whatever labels are inferred for the unlabeled training data.

We describe two ways to implement self-ensembling, Π -model and temporal ensembling. Both approaches surpass prior state-of-the-art results in semi-supervised learning by a considerable margin. We furthermore observe that self-ensembling improves the classification accuracy in fully labeled cases as well, and provides tolerance against incorrect labels.

Our Π -model can be seen as a simplification of the Γ -model of the ladder network by Rasmus et al. (2015), a previously presented network architecture for semi-supervised learning. Our temporal ensembling model also has connections to the bootstrapping method of Reed et al. (2014) targeted for training with noisy labels.

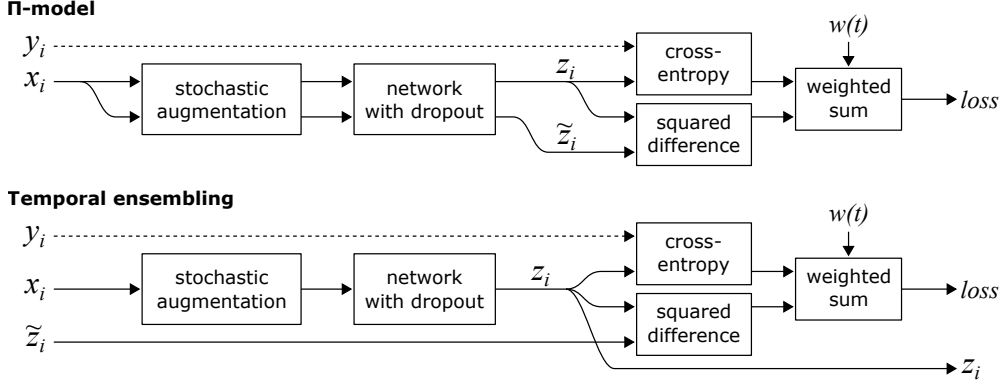


Figure 1: Structure of the training pass in our methods. Top: Π -model. Bottom: temporal ensembling. Labels y_i are available only for the labeled inputs, and the associated cross-entropy loss component is evaluated only for those.

Algorithm 1 Π -model pseudocode.

Require: x_i = training stimuli
Require: L = set of training input indices with known labels
Require: y_i = labels for labeled inputs $i \in L$
Require: $w(t)$ = unsupervised weight ramp-up function
Require: $f_\theta(x)$ = stochastic neural network with trainable parameters θ
Require: $g(x)$ = stochastic input augmentation function

```

for  $t$  in  $[1, num\_epochs]$  do
  for each minibatch  $B$  do
     $z_{i \in B} \leftarrow f_\theta(g(x_{i \in B}))$            ▷ evaluate network outputs for augmented inputs
     $\tilde{z}_{i \in B} \leftarrow f_\theta(g(x_{i \in B}))$        ▷ again, with different dropout and augmentation
     $loss \leftarrow -\frac{1}{|B|} \sum_{i \in (B \cap L)} \log z_i[y_i]$   ▷ supervised loss component
     $\quad + w(t) \frac{1}{C|B|} \sum_{i \in B} \|z_i - \tilde{z}_i\|^2$   ▷ unsupervised loss component
    update  $\theta$  using, e.g., ADAM                ▷ update network parameters
  end for
end for
return  $\theta$ 

```

2 SELF-ENSEMBLING DURING TRAINING

We present two implementations of self-ensembling during training. The first one, Π -model, encourages consistent network output between two realizations of the same input stimulus, under two different dropout conditions. The second method, temporal ensembling, simplifies and extends this by taking into account the network predictions over multiple previous training epochs.

We shall describe our methods in the context of traditional image classification networks. Let the training data consist of total of N inputs, out of which M are labeled. The input stimuli, available for all training data, are denoted x_i , where $i \in \{1 \dots N\}$. Let set L contain the indices of the labeled inputs, $|L| = M$. For every $i \in L$, we have a known correct label $y_i \in \{1 \dots C\}$, where C is the number of different classes.

2.1 Π -MODEL

The structure of Π -model is shown in Figure 1 (top), and the pseudocode in Algorithm 1. During training, we evaluate the network for each training input x_i twice, resulting in prediction vectors z_i and \tilde{z}_i . Our loss function consists of two components. The first component is the standard cross-entropy loss, evaluated for labeled inputs only. The second component, evaluated for all inputs, penalizes different predictions for the same training input x_i by taking the mean square difference

between the prediction vectors z_i and \tilde{z}_i .¹ To combine the supervised and unsupervised loss terms, we scale the latter by time-dependent weighting function $w(t)$. By comparing the entire output vectors z_i and \tilde{z}_i , we effectively ask the “dark knowledge” (Hinton et al., 2015) between the two evaluations to be close, which is a much stronger requirement compared to asking that only the final classification remains the same, which is what happens in traditional training.

It is important to notice that, because of dropout regularization, the network output during training is a stochastic variable. Thus two evaluations of the same input x_i under same network weights θ yield different results. In addition, Gaussian noise and augmentations such as random translation are evaluated twice, resulting in additional variation. The combination of these effects explains the difference between the prediction vectors z_i and \tilde{z}_i . This difference can be seen as an error in classification, given that the original input x_i was the same, and thus minimizing it is a reasonable goal.

In our implementation, the unsupervised loss weighting function $w(t)$ ramps up, starting from zero, during many training epochs. In the beginning the total loss and the learning gradients are thus dominated by the supervised loss component, i.e., the labeled data only. We have found it to be very important that the ramp-up of the unsupervised loss component is slow enough—otherwise, the network gets easily stuck in a degenerate solution where no meaningful classification of the data is obtained. The training parameters are discussed further in Appendix A.

Our approach is somewhat similar to the Γ -model of the ladder network by Rasmus et al. (2015), but conceptually simpler. In the Π -model, the comparison is done directly on network outputs, i.e., after softmax activation, and there is no auxiliary mapping between the two branches such as the learned denoising functions in the ladder network architecture. Furthermore, instead of having one “clean” and one “corrupted” branch as in Γ -model, we apply equal augmentation and noise to the inputs for both branches.

As shown in Section 3, the Π -model combined with a good convolutional network architecture provides a significant improvement over prior art in classification accuracy.

2.2 TEMPORAL ENSEMBLING

Analyzing how the Π -model works, we could equally well split the evaluation of the two branches in two separate phases: first classifying the training set once without updating the weights θ , and then training the network on the same inputs under different augmentations and dropout, using the just obtained predictions as targets for the unsupervised loss component. As the training targets obtained this way are based on a single evaluation of the network, they can be expected to be noisy. Temporal ensembling alleviates this by aggregating the predictions of multiple previous network evaluations into an ensemble prediction. It also lets us evaluate the network only once during training, gaining an approximate 2x speedup over the Π -model.

The structure of our temporal ensembling method is shown in Figure 1 (bottom), and the pseudocode in Algorithm 2. The main difference to the Π -model is that the network and augmentations are evaluated only once per input per epoch, and the target vectors \tilde{z} for the unsupervised loss component are based on prior network evaluations instead of a second evaluation of the network.

After every training epoch, the network outputs z_i are accumulated into ensemble outputs Z_i by updating $Z_i \leftarrow \alpha Z_i + (1 - \alpha)z_i$, where α is a momentum term that controls how far the ensemble reaches into training history. Because of dropout regularization and stochastic augmentation, Z thus contains a weighted average of the outputs of an ensemble of networks f from previous training epochs, with recent epochs having larger weight than distant epochs. For generating the training targets \tilde{z} , we need to correct for the startup bias in Z by dividing by factor $(1 - \alpha^t)$. A similar bias correction has been used in, e.g., Adam (Kingma and Ba, 2014) and mean-only batch normalization (Salimans and Kingma, 2016). On the first training epoch, Z and \tilde{z} are zero as no data from previous epochs is available. For this reason, we specify the unsupervised weight ramp-up function $w(t)$ to also be zero on the first training epoch.

The benefits of temporal ensembling compared to Π -model are twofold. First, the training is faster because the network is evaluated only once per input on each epoch. Second, the training targets

¹Squared difference gave slightly but consistently better results than cross-entropy loss in our tests.

Algorithm 2 Temporal ensembling pseudocode. Note that the updates of Z and \tilde{z} could equally well be done inside the minibatch loop; in this pseudocode they occur between epochs for clarity.

Require: x_i = training stimuli
Require: L = set of training input indices with known labels
Require: y_i = labels for labeled inputs $i \in L$
Require: α = ensembling momentum, $0 \leq \alpha < 1$
Require: $w(t)$ = unsupervised weight ramp-up function
Require: $f_\theta(x)$ = stochastic neural network with trainable parameters θ
Require: $g(x)$ = stochastic input augmentation function

```

 $Z \leftarrow \mathbf{0}_{[N \times C]}$                                 ▷ initialize ensemble predictions
 $\tilde{z} \leftarrow \mathbf{0}_{[N \times C]}$                                 ▷ initialize target vectors
for  $t$  in  $[1, num\_epochs]$  do
  for each minibatch  $B$  do
     $z_{i \in B} \leftarrow f_\theta(g(x_{i \in B}, t))$                 ▷ evaluate network outputs for augmented inputs
     $loss \leftarrow -\frac{1}{|B|} \sum_{i \in (B \cap L)} \log z_i[y_i]$         ▷ supervised loss component
     $+ w(t) \frac{1}{C|B|} \sum_{i \in B} ||z_i - \tilde{z}_i||^2$             ▷ unsupervised loss component
    update  $\theta$  using, e.g., ADAM                            ▷ update network parameters
  end for
   $Z \leftarrow \alpha Z + (1 - \alpha)z$                             ▷ accumulate ensemble predictions
   $\tilde{z} \leftarrow Z / (1 - \alpha^t)$                             ▷ construct target vectors by bias correction
end for
return  $\theta$ 

```

\tilde{z} can be expected to be less noisy than with Π -model. As shown in Section 3, we indeed obtain somewhat better results with temporal ensembling than with Π -model in the same number of training epochs. The downside compared to Π -model is the need to store auxiliary data across epochs, and the new hyperparameter α .

An intriguing additional possibility of temporal ensembling is collecting other statistics from the network predictions z_i besides the mean. For example, by tracking the second raw moment of the network outputs, we can estimate the variance of each output component $z_{i,j}$. This makes it possible to reason about the uncertainty of network outputs in a principled way (Gal and Ghahramani, 2016). Based on this information, we could, e.g., place more weight on more certain predictions vs. uncertain ones in the unsupervised loss term. However, we leave the exploration of these avenues as future work.

3 RESULTS

Our network structure is given in Table 3, and the test setup and all training parameters are detailed in Appendix A. We test the Π -model and temporal ensembling in two image classification tasks, CIFAR-10 and SVHN, and report the mean and standard deviation of 10 runs using different random seeds.

Although it is rarely stated explicitly, we believe that our comparison methods do not use input augmentation, i.e., are limited to dropout and other forms of permutation-invariant noise. Therefore we report the error rates without augmentation, unless explicitly stated otherwise. Given that the ability of an algorithm to extract benefit from augmentation is also an important property, we report the classification accuracy using a standard set of augmentations as well. In purely supervised training the de facto standard way of augmenting the CIFAR-10 dataset includes horizontal flips and random translations, while SVHN is limited to random translations. By using these same augmentations we can compare against the best fully supervised results as well. After all, the fully supervised results should indicate the upper bound of obtainable accuracy.

Table 1: CIFAR-10 results with 4000 labels, averages of 10 runs (4 runs for all labels).

	Error rate (%) with # labels	
	4000	All (50000)
Supervised-only	35.56 ± 1.59	7.33 ± 0.04
with augmentation	34.85 ± 1.65	6.05 ± 0.15
Conv-Large, Γ -model (Rasmus et al., 2015)	20.40 ± 0.47	
CatGAN (Springenberg, 2016)	19.58 ± 0.58	
GAN of Salimans et al. (2016)	18.63 ± 2.32	
Π -model	16.55 ± 0.29	6.90 ± 0.07
Π -model with augmentation	12.36 ± 0.31	5.56 ± 0.10
Temporal ensembling with augmentation	12.16 ± 0.24	5.60 ± 0.10

Table 2: SVHN results for 500 and 1000 labels, averages of 10 runs (4 runs for all labels).

Model	Error rate (%) with # labels		
	500	1000	All (73257)
Supervised-only	35.18 ± 5.61	20.47 ± 2.64	3.05 ± 0.07
with augmentation	31.59 ± 3.60	19.30 ± 3.89	2.88 ± 0.03
DGN (Kingma et al., 2014)		36.02 ± 0.10	
Virtual Adversarial (Miyato et al., 2016)		24.63	
ADGM (Maaløe et al., 2016)		22.86	
SDGM (Maaløe et al., 2016)		16.61 ± 0.24	
GAN of Salimans et al. (2016)	18.44 ± 4.8	8.11 ± 1.3	
Π -model	7.05 ± 0.30	5.43 ± 0.25	2.78 ± 0.03
Π -model with augmentation	6.65 ± 0.53	4.82 ± 0.17	2.54 ± 0.04
Temporal ensembling with augmentation	5.12 ± 0.13	4.42 ± 0.16	2.74 ± 0.06

3.1 CIFAR-10

CIFAR-10 is a dataset consisting of 32×32 pixel RGB images from ten classes. Table 1 shows a 2.1 percentage point reduction in classification error rate with 4000 labels (400 per class) compared to earlier methods for the non-augmented Π -model.

Enabling the standard set of augmentations further reduces the error rate by 4.2 percentage points to 12.36%. Temporal ensembling is slightly better still at 12.16%, while being twice as fast to train. This small improvement conceals the subtle fact that random horizontal flips need to be done independently for each epoch in temporal ensembling, while Π -model can randomize once per a pair of evaluations, which according to our measurements is ~ 0.5 percentage points better than independent flips.

3.2 SVHN

The street view house numbers (SVHN) dataset consists of 32×32 pixel RGB images of real-world house numbers, and the task is to classify the centermost digit. In SVHN we chose to use only the official 73257 training examples because otherwise the dataset would have been too easy. Even with this choice our error rate with all labels is only 3.05% without augmentation.

Table 2 compares our method to the previous state-of-the-art. With the most commonly used 1000 labels we observe an improvement of 2.7 percentage points, from 8.11% to 5.43% without augmentation, and further to 4.42% with standard augmentations.

We also investigated the behavior with 500 labels, where we obtained an error rate less than half of Salimans et al. (2016) without augmentations, with a significantly lower standard deviation as well. When augmentations were enabled, temporal ensembling further reduced the error rate to 5.12%. In this test the difference between Π -model and temporal ensembling was quite significant at 1.5 percentage points.

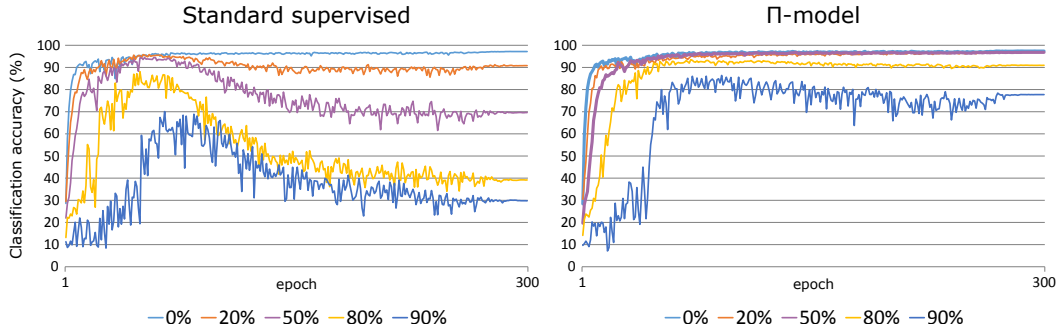


Figure 2: Percentage of correct SVHN classifications as a function of training epoch when a part of the labels is randomized. With standard supervised training (left) the classification accuracy suffers when even a small portion of the labels give disinformation, and the situation worsens quickly as the portion of randomized labels increases to 50% or more. On the other hand, Π -model (right) shows almost perfect resistance to disinformation when half of the labels are random, and retains over ninety percent classification accuracy even when 80% of the labels are random.

3.3 SUPERVISED LEARNING

When all labels are used for traditional supervised training, our network approximately matches the state-of-the-art error rate for a single model in CIFAR-10 with augmentation (Lee et al., 2015; Mishkin and Matas, 2016) at 6.05%, and without augmentation (Salimans and Kingma, 2016) at 7.33%. The same is probably true for SVHN as well, but there the best published results rely on extra data that we chose not to use.

Given this premise, it is perhaps somewhat surprising that our methods reduce the error rate also when all labels are used (Tables 1 and 2). To the best of our knowledge at least the CIFAR-10 score may be the best documented so far using a single model and standard augmentations. We believe that this is an indication that the consistency requirement adds a degree of resistance to ambiguous labels that are fairly common in many classification tasks. One way to interpret this is that our self-ensembling benefits from the “dark knowledge” (Hinton et al., 2015) present in network outputs.

3.4 TOLERANCE TO INCORRECT LABELS

In a further preliminary test we studied the hypothesis that our methods add tolerance to incorrect labels by assigning a random label to a certain percentage of the training set before starting to train. Figure 2 shows the classification error graphs for standard supervised training and the Π -model. Clearly the Π -model provides considerable resistance to wrong labels, which should be a useful property especially if it translates to regression problems.

4 RELATED WORK

There is a large body of previous work on semi-supervised learning (Zhu, 2005). In here we will concentrate on the ones that are most directly connected to our work.

Γ -model is a subset of a ladder network (Rasmus et al., 2015) that introduces lateral connections into an encoder-decoder type network architecture, targeted at semi-supervised learning. In Γ -model, all but the highest lateral connections in the ladder network are removed, and after pruning the unnecessary stages, the remaining network consists of two parallel, identical branches. One of the branches takes the original training inputs, whereas the other branch is given the same input corrupted with noise. The unsupervised loss term is computed as the squared difference between the (pre-activation) output of the clean branch and a denoised (pre-activation) output of the corrupted branch. The denoised estimate is computed from the output of the corrupted branch using a parametric nonlinearity that has 10 auxiliary trainable parameters per unit. Our Π -model differs from the Γ -model in removing the parametric nonlinearity and denoising, having two corrupted paths, and comparing the outputs of the network instead of pre-activation data of the final layer.

Table 3: The network architecture used in all of our tests.

NAME	DESCRIPTION
input	32×32 RGB image
noise	Additive Gaussian noise $\sigma = 0.15$
conv1a	128 filters, 3×3 , pad = 'same', LReLU ($\alpha = 0.1$)
conv1b	128 filters, 3×3 , pad = 'same', LReLU ($\alpha = 0.1$)
conv1c	128 filters, 3×3 , pad = 'same', LReLU ($\alpha = 0.1$)
pool1	Maxpool 2×2 pixels
drop1	Dropout, $p = 0.5$
conv2a	256 filters, 3×3 , pad = 'same', LReLU ($\alpha = 0.1$)
conv2b	256 filters, 3×3 , pad = 'same', LReLU ($\alpha = 0.1$)
conv2c	256 filters, 3×3 , pad = 'same', LReLU ($\alpha = 0.1$)
pool2	Maxpool 2×2 pixels
drop2	Dropout, $p = 0.5$
conv3a	512 filters, 3×3 , pad = 'valid', LReLU ($\alpha = 0.1$)
conv3b	256 filters, 1×1 , LReLU ($\alpha = 0.1$)
conv3c	128 filters, 1×1 , LReLU ($\alpha = 0.1$)
pool3	Global average pool ($6 \times 6 \rightarrow 1 \times 1$ pixels)
dense	Fully connected $128 \rightarrow 10$
output	Softmax

In bootstrap aggregating, or *bagging*, multiple networks are trained independently based on subsets of training data (Breiman, 1996). This results in an ensemble that is more stable and accurate than the individual networks. Our approach can be seen as pulling the predictions from an implicit ensemble that is based on a single network, and the variability is a result of evaluating it under different dropout and augmentation conditions instead of training on different subsets of data.

The general technique of inferring new labels from partially labeled data is often referred to as *bootstrapping* or *self-training*, and it was first proposed by Yarowsky (1995) in the context of linguistic analysis. Whitney and Sarkar (2012) analyze Yarowsky’s algorithm and propose a novel graph-based label propagation approach. Similarly, label propagation methods (Zhu and Ghahramani, 2002) infer labels for unlabeled training data by comparing the associated inputs to labeled training inputs using a suitable distance metric. Our approach differs from this in two important ways. Firstly, we never compare training inputs against each other, but instead only rely on the unknown labels remaining constant, and secondly, we let the network produce the likely classifications for the unlabeled inputs instead of providing them through an outside process.

In addition to partially labeled data, considerable amount of effort has been put into dealing with densely but inaccurately labeled data. This can be seen as a semi-supervised learning task where part of the training process is to identify the labels that are not to be trusted. For recent work in this area, see, e.g., Sukhbaatar et al. (2014) and Patrini et al. (2016). In this context of noisy labels, Reed et al. (2014) presented a simple bootstrapping method that trains a classifier with the target composed of a convex combination of the previous epoch output and the known but potentially noisy labels. Our temporal ensembling differs from this by taking into account the evaluations over multiple previous epochs.

Generative Adversarial Networks (GAN) have been recently used for semi-supervised learning with promising results (Maaløe et al., 2016; Springenberg, 2016; Odena, 2016; Salimans et al., 2016). It could be an interesting avenue for future work to incorporate a generative component to our solution. We also envision that our methods could be applied to regression-type learning tasks.

REFERENCES

- L. Breiman. Bagging predictors. *Machine Learning*, 24(2), 1996.
- S. Dieleman, J. Schlüter, C. Raffel, E. Olson, S. K. Sønderby, et al. Lasagne: First release., 2015.
- Y. Gal and Z. Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. *CoRR*, abs/1506.02142, 2016.

- K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, abs/1502.01852, 2015.
- G. E. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *CoRR*, abs/1503.02531, 2015.
- G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger. Deep networks with stochastic depth. *CoRR*, abs/1603.09382, 2016.
- D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.
- D. P. Kingma, S. Mohamed, D. Jimenez Rezende, and M. Welling. Semi-supervised learning with deep generative models. In *Advances in Neural Information Processing Systems 27 (NIPS)*. 2014.
- C.-Y. Lee, P. W. Gallagher, and Z. Tu. Generalizing pooling functions in convolutional neural networks: Mixed, gated, and tree. *CoRR*, abs/1509.08985, 2015.
- L. Maaløe, C. K. Sønderby, S. K. Sønderby, and O. Winther. Auxiliary deep generative models. *CoRR*, abs/1602.05473, 2016.
- A. L. Maas, A. Y. Hannun, and A. Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. International Conference on Machine Learning (ICML)*, volume 30, 2013.
- D. Mishkin and J. Matas. All you need is a good init. In *Proc. International Conference on Learning Representations (ICLR)*, 2016.
- T. Miyato, S. Maeda, M. Koyama, K. Nakae, and S. Ishii. Distributional smoothing with virtual adversarial training. In *Proc. International Conference on Learning Representations (ICLR)*, 2016.
- A. Odena. Semi-supervised learning with generative adversarial networks. *Data Efficient Machine Learning workshop at ICML 2016*, 2016.
- G. Patrini, A. Rozza, A. Menon, R. Nock, and L. Qu. Making neural networks robust to label noise: a loss correction approach. *CoRR*, abs/1609.03683, 2016.
- A. Rasmus, M. Berglund, M. Honkala, H. Valpola, and T. Raiko. Semi-supervised learning with ladder networks. In *Advances in Neural Information Processing Systems 28 (NIPS)*. 2015.
- S. E. Reed, H. Lee, D. Anguelov, C. Szegedy, D. Erhan, and A. Rabinovich. Training deep neural networks on noisy labels with bootstrapping. *CoRR*, abs/1412.6596, 2014.
- T. Salimans and D. P. Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *CoRR*, abs/1602.07868, 2016.
- T. Salimans, I. J. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. Improved techniques for training GANs. *CoRR*, abs/1606.03498, 2016.
- S. Singh, D. Hoiem, and D. A. Forsyth. Swapout: Learning an ensemble of deep architectures. *CoRR*, abs/1605.06465, 2016.
- J. T. Springenberg. Unsupervised and semi-supervised learning with categorical generative adversarial networks. In *Proc. International Conference on Learning Representations (ICLR)*, 2016.
- J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. A. Riedmiller. Striving for simplicity: The all convolutional net. *CoRR*, abs/1412.6806, 2014.
- N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.
- S. Sukhbaatar, J. Bruna, M. Paluri, L. Bourdev, and R. Fergus. Training convolutional networks with noisy labels. *CoRR*, abs/1406.2080, 2014.
- Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *CoRR*, abs/1605.02688, May 2016.
- L. Wan, M. Zeiler, S. Zhang, Y. L. Cun, and R. Fergus. Regularization of neural networks using dropconnect. *Proc. International Conference on Machine Learning (ICML)*, 28(3):1058–1066, 2013.
- M. Whitney and A. Sarkar. Bootstrapping via graph propagation. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers - Volume 1*, ACL ’12, 2012.

- D. Yarowsky. Unsupervised word sense disambiguation rivaling supervised methods. In *Proceedings of the 33rd Annual Meeting on Association for Computational Linguistics, ACL '95*, 1995.
- X. Zhu. Semi-supervised learning literature survey. Technical Report 1530, Computer Sciences, University of Wisconsin-Madison, 2005.
- X. Zhu and Z. Ghahramani. Learning from labeled and unlabeled data with label propagation. Technical Report CMU-CALD-02-107, Carnegie Mellon University, 2002.

A NETWORK ARCHITECTURE, TEST SETUP, AND TRAINING PARAMETERS

Table 3 details the network architecture used in all of our tests. It is heavily inspired by ConvPool-CNN-C (Springenberg et al., 2014) and the improvements made by Salimans and Kingma (2016). All data layers were initialized following He et al. (2015), and we applied weight normalization and mean-only batch normalization (Salimans and Kingma, 2016) with momentum 0.999 to all of them. We used leaky ReLU (Maas et al., 2013) with $\alpha = 0.1$ as the non-linearity, and chose to use max pooling instead of strided convolutions because it gave consistently better results in our experiments.

All networks were trained using Adam (Kingma and Ba, 2014) with a maximum learning rate of $\lambda_{max} = 0.003$, except for temporal ensembling in the SVHN case where a maximum learning rate of $\lambda_{max} = 0.001$ worked better. Adam momentum parameters were set to $\beta_1 = 0.9$ and $\beta_2 = 0.999$ as suggested in the paper. The maximum value for the unsupervised loss component was set to $w_{max} \cdot M/N$, where M is the number of labeled inputs and N is the total number of training inputs. For Π -model runs, we used $w_{max} = 100$, except for the random label test in Section 3.4 where $w_{max} = 1000$ gave better results. In all temporal ensembling runs we used $w_{max} = 30$, and the accumulation decay constant was set to $\alpha = 0.6$.

In all runs we ramped up both the learning rate λ and unsupervised loss component weight w during the first 80 epochs using a Gaussian ramp-up curve $\exp[-5(1 - T)^2]$, where T advances linearly from zero to one during the ramp-up period. In addition to ramp-up, we annealed the learning rate λ to zero and Adam β_1 to 0.5 during the last 50 epochs, but otherwise we did not decay them during training. The ramp-down curve was similar to the ramp-up curve but time-reversed and with a scaling constant of 12.5 instead of 5. All networks were trained for 300 epochs with minibatch size of 100.

CIFAR-10 Following previous work in fully supervised learning, we pre-processed the images using ZCA and augmented the dataset using horizontal flips and random translations. The translations were drawn from $[-2, 2]$ pixels, and were independently applied to both branches in the Π -model.

SVHN We pre-processed the input images by biasing and scaling each input image to zero mean and unit variance. We used only the 73257 items in the official training set, i.e., did not use the provided 531131 extra items. The training setups were otherwise similar to CIFAR-10 except that horizontal flips were not used.

Implementation Our implementation is written in Python using Theano (Theano Development Team, 2016) and Lasagne (Dieleman et al., 2015), and is available at <https://github.com/smlaine2/tempens>.

Model convergence As discussed in Section 2.1, a slow ramp-up of the unsupervised cost is very important for getting the models to converge. Furthermore, in our very preliminary tests with 250 labels in SVHN we noticed that optimization tended to explode during the ramp-up period, and we eventually found that using a lower value for Adam β_2 parameter (e.g., 0.99 instead of 0.999) seems to help in this regard. The exact reason is not presently understood, but this observation may be useful if probing the limits of our methods using the provided implementation.

We do not attempt to guarantee that the occurrence of labeled inputs during training would be somehow stratified; with bad luck there might be several consecutive minibatches without any labeled inputs when the label density is very low. Some previous work has identified this as a weakness, and have solved the issue by shuffling the input sequences in such a way that stratification is guaranteed, e.g. Rasmus et al. (2015) (confirmed from the authors). This kind of stratification might further improve the convergence of our methods as well.